Journal of Applied Sciences 6 (15): 3122-3127, 2006
ISSN 1812-5654
© 2006 Asian Network for Scientific Information

A Prefetching Algorithm for Improving Web Cache Performance

[1]C. Umapathi and [2]J. Raja
[1]Sathyabama Institute of Science and Technology, Chennai-119 India
[2]S.S.N College of Engineering, Chennai-110 India

Abstract: Predictive Web pre fetching refers to the mechanism of deducing the forthcoming page accesses of a client based on its past accesses. Nowadays, improving the performance of the Web is a crucial requirement, since its popularity resulted in a large increase in the user perceived latency. In this study, we describe a Web pre fetching cum caching scheme that capitalizes on Web log mining methods. The proposed scheme achieves coordination between the two techniques (i.e., caching and pre fetching). Here the pre fetched documents are accommodated in a dedicated part of the cache, to avoid the drawback of incorrect replacement of requested documents. The requirements of the Web are taken into account, compared to the existing schemes for buffer management in database and operating systems. Experimental results indicate the superiority of the proposed method compared to the previous ones, in terms of improvement in cache performance.

Key words: Prediction, pre fetching, web caching, apriori, web log mining, association rules

INTRODUCTION

In recent years, the Web has become the primary means for information dissemination. It is being used for commercial, entertainment, or educational purposes and, thus, its popularity resulted in heavy traffic in the Internet. Since the Internet capacity is not keeping pace, the net effect of this growth was a significant increase in the user perceived latency, that is, the time between when a client issues a request for a document and the time the response arrives. Potential sources of latency are the Web servers' heavy load, network congestion, low bandwidth, band width underutilization and propagation delay.

The problem of modeling and predicting a user's accesses on a Web-site has attracted a lot of research interest. It has been used to improve the Web performance through caching and prefetching, recommend related pages, improve search engines and personalize browsing in a Web site.

The caching of Web documents at various points in the network (client, proxy and server) has been developed to reduce latency. Caching capitalizes on the temporal locality. Effective client and proxy caches reduce the client perceived latency, the server load and the number of traveling packets, thus increase the available bandwidth. Nevertheless there exist cases where the benefits reaped due to caching can be limited, e.g., when Web resources tend to change very frequently, resources cannot be cached (dynamically generated Web documents), when they contain cookies (this issue matters only caching proxies),or when request streams do not exhibit high temporal locality.

Motivation: Web pre fetching is the process of deducing client's future requests for Web documents and getting that document into the cache, in the background, before an explicit request is made for them. Prefetching capitalizes on the spatial locality present in request streams, that is, correlated references for different documents and exploits the client's idle time, i.e., the time between successive requests. The main advantage of employing prefetching is that it prevents band-width underutilization and hides part of the latency. Additionally, without a carefully designed prefetching scheme, several transferred documents may not be used by the client at all, thus they waste bandwidth. We focus on predictive prefetching. Web prefetching acts complementary to caching, it can significantly improve cache performance and reduce the user-perceived latency .However, there are cases where a non-effective prefetching algorithm, presenting the aforementioned drawbacks, can impact cache performance For instance, if the accuracy of the prefetching algorithm is low, then several useful documents in the cache may be evicted by prefetched documents that are not going to be referenced. Therefore, there exists a requirement for both accurate prefetching algorithms and caching schemes that will coordinate with prefetching.

Paper contribution: In this study we describe a scheme for: (a) Effective prefetching, which exploits Web log mining and it is not affected by factors like noise (i.e., random document requests) and high-order dependencies among document requests and thus, can significantly improve cache performance. (b) Coordination of caching

Corresponding Author: C. Umapathi, Sathyabama Institute of Science and Technology, Chennai-119 India

and prefetching, by storing in the cache the prefetched documents separately from those which have been explicitly requested. The latter approach is based on the one of, which dedicates part of the cache to separately accommodate prefetched documents. Experimental results indicate that the proposed scheme outperforms existing ones in terms of improvement in cache performance.

RELATED WORK

In general, there exist two pre fetching approaches. The first approach is characterized as informed Pre fetching and the second approach is called predictive pre fetching, which is more viable. Research on predictive Web prefetching has involved the significant issue of log file processing and the determination of user transactions (sessions) from it. However, the most important factor in Web prefetching is the prediction algorithm. For the purpose of prediction, most of the Web prefetching schemes relies on existing algorithms from the context of file systems. This approach neglects issues that arise in the case of Web and stem from both the contents of Web documents in a site (which induce dependencies to their references) and the site's structure, i.e., the links among documents (which affect user 's navigation).

The scheme described by Padmanabhan and Mogul (1996) uses a prefetching algorithm proposed in the context of file systems. It constructs a data structure, called the Dependency Graph (DG), which maintains the pattern of access to different documents store at the server. As described above, the choice of forthcoming pages can depend, in general, on a number of previously visited pages. DG considers only first order dependencies. Thus, if several previous visits have to be considered (i.e., high-order dependencies), DG does not take them into account. The work described by Bestavros (1996), uses essentially the approach of dependency graph, but it makes predictions by computing the transitive closure of this graph. This method did not show significantly better results compared to the simple Dependency graph. Also the pre fetching algorithm which used the context of file systems called an m-order Prediction-by-Partial-Match (PPM) predictor did not show good results (Deshpande and Karypis, 2001). During a session, although a user may navigate according to a pattern's he may also randomly navigate to pages that do not belong to any pattern (and can be modeled as noise).Hence, a session can both contain documents belonging to patterns and others that do not and these documents are interleaved. However, PPM considers only subsequences of consecutive documents inside sessions, thus it is affected by the existence of noise. Moreover and PPM uses a constant maximum value for the order. However, no method for the determination of this value is provided. (Fan, 1999; Palpanas and Mendelzon, 1999) A choice of a small maximum may have a similar disadvantage as in the case of DG, whereas a choice of a large maximum may lead to unnecessary computational cost, due to maintenance of a large number of rules. The Web prefetching strategy, proposed by Lan *et al.* (1999), develops a specialized association rule mining algorithm to discover the prefetched documents. It discovers dependencies between pairs of documents (association rules with one item in the head and one item in the body). However, this scheme, similar to DG, considers only first order dependencies and, similar to PPM, it considers only consecutive subsequences within sessions. The improvement of the efficiency of PPM is examined, based on three pruning criteria. These criteria are used in a post-processing step, on the set of discovered rules and can be applied to any prefetching scheme, thus they are orthogonal issues to the subject examined in this study.

Web caching has received significant attention and several new algorithms were proposed, ranging from extensions to traditional policies (like LRU, LFU, etc.) to key-based policies and more sophisticated function-based policies, such as GD-Size, PSS. Moreover, significant results regarding optimal on-line and off-line caching policies for the Web were presented. Regarding the coordination of caching and prefetching, Jeon and Noh presented the W^2R algorithm. Motivated by the 2Q algorithm (Johnson and Shasha, 1994), W^2 R divides the available cache space into two partitions, called Weighing Room and Waiting Room. Prefetched documents initially enter the Waiting Room, before becoming normal cached documents in the Weighing Room. However, W2R was designed for database disk buffer management. It uses the One Block Lookahead (OBL) prefetching algorithm, which prefetches only one page each time. Moreover, all pages are of the same size. Web caching presents significantly different requirements, since different prefetching algorithms than the simple OBL are used (several documents are allowed to be prefetched each time and thus we must prioritize among them) and documents of different sizes have to be accommodated in the cache.

MECHANISM OF PREDICTIVE PRE FETCHING

Deduction of future references on the basis of predictive pre fetching can be implemented by having an engine which, after processing the past references, derives the probability of future access for the documents accessed so far. The prediction engine can reside either in the client or in the server side. In the former case, it uses the set of past references to find correlations and initiates pre fetching. No modifications need to be made neither to the current Web infrastructure (e.g., HTTP protocol, Web servers) nor to Web browsers if the pre fetcher module runs as a proxy in the browser (Hosseini-Khayat, 2000). The main limitation of this approach is that the clients, in general, lack sufficient information to discover the correlations between documents since their Requests cover a broad range of Web servers and an even broader range of documents. On the other hand, Web servers are in better position to make predictions about future references since they log a significant part of requests by all Internet clients for the resources they own.

The main drawback of the latter approach is that additional communication between the server and the client is needed in order to realize the pre fetching scheme. This scheme can be implemented by either the dissemination of predicted resources to the client [Barford and Crorella, 1998) or exchange of messages between server and clients, having the server piggybacking information about the predicted resources onto regular response messages, avoiding establishment of any new TCP connections (Brin and Page, 1998). Such a mechanism has been implemented in (Brin and Page, 1998; Curewitz *et al.*, 1993) and seems the most appropriate since it requires relatively few enhancements to the current request-response protocol and no changes to the HTTP 1.1 protocol.

Therefore, we assume that there is a system implementing a server-based predictive pre fetcher, which applies the Pre fetch procedure (Fig. 1). The server piggybacks its predictions to the clients only as hints (in this case, the prefetchSeq in Pre fetch procedure comprises these hints, i.e., ID numbers of the documents to be pre fetched). The client receives these hints and discards all those which correspond to documents found in its cache. Then, in a second stage, pre fetching takes place requesting the predicted documents. The caching of the requested documents (on-demand and pre fetched) is performed with the PECache procedure.

PREFETCHING ALGORITHM

None of the existing prefetching algorithms addresses at the same time both the factors of noise and high-order dependencies, that may exist within transactions (i.e., user sessions). In this section we describe an algorithm that addresses all the aforementioned factors. It uses the history of user accesses, maintained in the Web server's log file, to derive rules. Since the rules, which are appropriate for the prefetching, should be based on the navigation behavior of the client (expressed as the process of visiting links) we describe a pruning criterion that is based on the site structure. This pruning can significantly reduce the computational overhead. Prefetching is performed with the procedure depicted in Fig. 2. In this algorithm, R denotes the current request stream formed by the user and M is the maximum number of prefetched documents (user parameter). Also, we use an upper limit, called maxSize, in the size of each prefetched document, since it is not desired to transfer very large documents to avoid waste of bandwidth in case of an incorrect prediction.

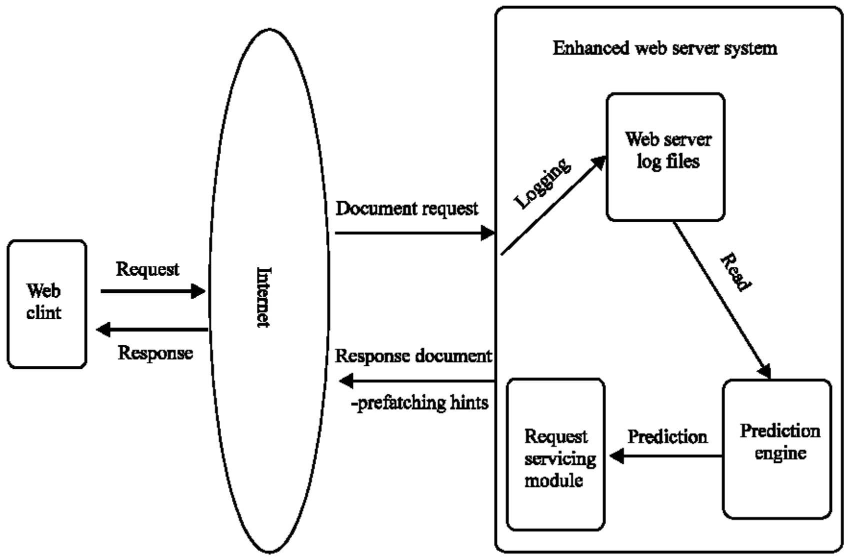

Fig. 1: Proposed architecture of a prediction-enabled web server

```
Procedure Prefetch (Array R, int M, float maxSize)
//h, b are sequences of document ids
begin
1. PrefetchSeq = Ø
2. for each rule h → b such that h ≤ R
3. for each d ▯ b such that d.size < maxSize
4. PrefetchSeq = prefetchSeq U d
5. end for
6. endfor
7. Sort documents in prefetchSeq in decreasing order of
   the confidence Of the corresponding rule and keep the
   first M ones.
8. Return prefetchSeq
end
```

Fig. 2: Prefetching algorithm

Evidently, the fact that the ordering of documents within transactions is preserved during the discovery of rules, impacts the complexity of candidate generation and support counting procedures. For this reason, we present in the following section a pruning criterion according to the site structure that reduces the overhead.

PRUNING CRITERION

For the purposes of prefetching, we focus on the paradigm of traversal patterns, which will be used for prediction, not on usage patterns. Based on the assumption that navigation is performed by following the hypertext links, the traversal patterns have to reflect the way navigation is performed guided by the site structure. Thus, we can apply pruning according to the structure of the site.

Since the pruning is based on the structure of the site, results in a significant reduction in the number of candidates. Without pruning, due to the consideration of ordering, a large number of candidates would have been generated. According to the proposed pruning criterion, (we take the candidate generation algorithm in (Alexandros *et al*., 2003) an access sequence and thus a candidate, has to correspond to a path in this graph. The candidate generation procedure and the apriori-pruning criterion have to be modified appropriately.

THE CACHING ALGORITHM

The caching algorithm PECache proposed here, divides the cache into a Weighing Room (uses LRU as the replacement policy) and a Waiting Room (uses the FIFO policy). This division of the cache space aims at isolating the effect of document mispredictions or the effect of aggressive pre fetching. It achieves this by dedicating part of the cache space to exploit the temporal locality of the request stream (on-demand requests) and the rest of the cache space is dedicated to exploit the spatial locality (prefetch requests). The relative size of the partitions should reflect the amount and type of the locality of the request stream.

The caching procedure PECache (Prefetch Enhanced Cache), given in Fig. 3, has as input the requested document (d) and the current request stream of the user (R). The PECache procedure uses the prefetching algorithm (step 3) that may return several documents, whereas W^2R uses the OBL prefetching algorithm, which always prefetches one document. Therefore, differently from the W^2R algorithm, the set of prefetched documents are inserted in the FIFO structure of the Waiting Room according to the corresponding confidence values (this is performed at step 5 of the PECache procedure). It is assumed that at steps 4-5 of the PECache procedure the prefetched documents enter the FIFO structure in the exact order they were requested, i.e., the caching mechanism resolves the issues of identifying the documents that belong to the same prefetchSeq and sorting them according to the requested order. Moreover, differently from W^2R, the PECache procedure does not perform prefetching in the case the requested document d, is contained in the Waiting Room (step 7). Otherwise, this would result in excessive network traffic and bandwidth consumption (notice that W^2R is designed for buffer management in a DBMS). It should be mentioned that the replacement policy used in the Weighing Room can be selected independently.

```
Procedure PECache(Array R, Document d)
begin
1. R = RUd
2. if not (d in Weighing Room or d in Waiting Room)
2. put d at head of the LRU list of the Weighing Room
3. prefetchSeq = Prefetch(R, M, maxSize)
4. foreach p in prefetchSeq
5. append p at the end of Waiting Room queue
6. endfor
7. else if d in Waiting Room
8. remove d from Waiting Room
9. put d at head of the LRU list of the Weighing Room
10. else if d in Weighing Room
11. put d at head of the LRU list of the Weighing Room
12. endif
end
```

Fig. 3: The caching procedure

PERFORMANCE RESULTS

This section presents the experimental results. The cache performance is examined against the factors of high-order dependencies, amount of noise and cache size. The performance measure is the hit ratio achieved by the cache. We examine the performance of the proposed caching policy, in coordination with the prefetching algorithm. The proposed method is denoted as PEC (Prefetch Enhanced Cache). For the purposes comparison, we also examine the cache performance in the case of using the DG, PPM and LBOT prefetching algorithms in coordination with the caching policy of Section 6, so as to clearly identify the advantages of the proposed prefetching algorithm. Additionally, we examine the performance of the plain LRU caching policy (i.e., when no prefetching is performed and only one cache partition is used), so as to identify the advantages of the proposed caching policy. In the case where one cache partition is used, its size is equal to the sum of sizes of the two partitions (i.e., Weighing and Waiting rooms) of the proposed caching policy. We separately examine the impact of the proposed pruning criterion on the reduction of the number of candidates during the generation of prefetching rules for the PEC method. In order to carry out the experiments we generated a number of workloads. Each workload consisted of T = 100,000 transactions. From these, 30,000 transactions were used to train the algorithms and the rest to evaluate their performance. The number of documents of the site for all workloads was fixed to N = 1000 and the maximum fan out to Nfanout = 100, so as to simulate a dense site. The branching factor was set to bf = 4 to simulate relatively low correlation between the paths. The number of paths of the pool for all workloads was fixed to P = 1000. With several experiments, not shown in this report, it was found that varying the values of the parameters P and N does not affect the relative performance of the considered algorithms. For all the experiments presented here, the order of the PPM algorithm was set equal to 5, so as to capture both low and higher order dependencies. The default value for the mean transaction size was set to 10. Throughout the experiments, the range of cache size was selected to be in the range of few hundred KB, to simulate the fact that not all, but only a small part of the Web client's cache is dedicated to the documents of a particular Web server. The confidence threshold was tuned separately for each algorithm, so as to derive the same network traffic overhead, which is defined to be the number of documents that the client gets when prefetching is used divided by the one when prefetching is not used. The examined network traffic was 150%. This is also the value of the average network byte overhead. This means that for each byte the user requested, the prefetchers fetched another 0.5 byte that the user never requested. This is a relatively conservative approach considering that existing techniques and implementations incur a much larger overhead. First, we evaluated the impact of varying order on the hit ratio (Notice that the order of dependencies varies with the type of the site). The mean noise value was set to 1.0. The total cache size was set to 150 and 50 KB of this total size were dedicated to the Waiting Room (this does not apply for the case of plain LRU). The results of this set of experiments are reported in Fig. 4.

Next, we assessed the impact of noise on the hit ratio. The order value was set to 0.5. The results of this set of experiments are reported in Fig. 5.

We measured the impact of cache size. We kept the Waiting Room size equal to 50 KB and varied the total cache size (which includes the size of the Waiting Room, besides the case of plain LRU caching policy). The mean noise value was set to 1.5 and the order was set to 0.5. Figure 6 illustrates the results for all methods.

As depicted, the hit ratio increases linearly with increasing cache size. PEC presents the best performance in all cases, whereas PPM the second best. As in the previous cases, plain LRU presents the worst hit ratio among all methods.

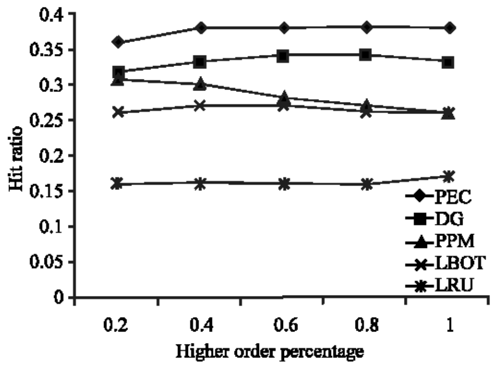

Fig. 4: Results on hit ratio w.r.t. order

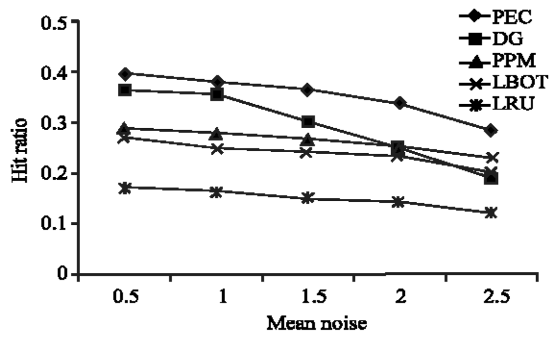

Fig. 5: Results on hit ratio w.r.t. noise

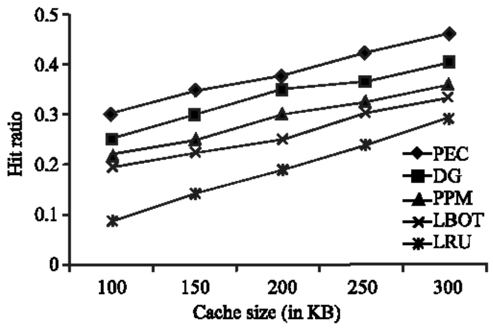

Fig. 6: Results on hit ratio w.r.t cache size

CONCLUSIONS

We considered the problem of enhancing Web caching with the technique of predictive Web prefetching. We also examined the problem of the coordination between Web caching and prefetching. We proposed a new algorithm called PEC, which focuses both on attaining accurate prefetching and using cooperative caching so as to effectively accommodate the prefetched documents with the normal cached ones (i.e., the ones cached after an explicit user request). For the former factor, we described a prefetching algorithm, which exploits Web log mining techniques. To address the problem of large computational overhead for the rule generation phase, we described a pruning criterion that is based on the site structure. For the latter factor, we presented an algorithm which uses a small part of the cache so as to separately store the prefetched documents. We addressed the new requirements due to the particularities of the Web Experimental results illustrated the superiority of PEC. In contrast to existing methods, PEC is not affected by factors like high-order dependencies among document references, or the existence of noise within user transactions. Also, experimental results showed the effectiveness of the pruning criterion.

The examination of other caching policies within the framework of PEC. And the development of dynamic methods for the tuning of the Waiting Room Size is our future work.

REFERENCES

Alexandros, N. *et al*., 2003. A data mining algorithm for generalized web prefetching. IEEE Transactions on Knowledge and Data Eng., 15: 1155-1169.

Barford, P. and M. Crovella, 1998. Generating representative Web workloads for network and server performance evaluation. In Proceedings of the ACM Conference on Measurement and Modeling of Computer Systems, (ACM SIGMETRICS'98), pp: 151-160.

Bestavros, A., 1996. Speculative data dissemination and service to reduce server load, network traffic and service time. In: Proceedings of the IEEE Conference on Data Engineering (ICDE'96), pp: 180-189.

Brin, S. and L. Page, 1998. The anatomy of large-scale hypertextual Web search engine. In: Proceedings of the World Wide Web Conference (WWW'98), pp: 107-117.

Curewitz, K.M., P. Krishnan and J.S. Vitter, 1993. Practical prefetching via data compression.In Proceedings of the ACM Conference on Management of Data (ACMSIGMOD'93), pp: 257-266.

Deshpande, M. and G. Karypis, 2001. Selective Markov models for predicting Web page accesses. In: Proceedings of the SIAM Conference on Data Mining (SDM'01).

Fan, L., P. Cao, W. Lin and Q. Jacobson, 1999. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In Proceedings of the ACM Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS' 99), pp: 178-187.

Hosseini-Khayat, S., 2000. On optimal replacement of nonuniform cache objects. IEEE Transactions on Computers, 49: 769-778.

Johnson, T. and D. Shasha, 1994. 2Q: A low overhead high performance buffer management replacement algorithm. In: Proceedings of the 20th Conference on Very Large Data Bases (VLDB'94), pp: 439-450.

Lan, B., S. Bressan, B.S. Ooi and Y. Tay, 1999. Making Web servers pushier. In Proceedings of the Workshop on Web Usage Analysis and User Profiling (WEBKDD'99).

Padmanabhan, V. and J. Mogul, 1996. Using predictive prefetching to improve World Wide Web latency. ACM SIGCOMM Computer Communications Review, 26(3).

Palpanas, T. and A. Mendelzon, 1999. Web prefetching using partial match prediction. In: Proceedings of the 4th Web Caching Workshop.